SCR*: A Toolset for Specifying and Analyzing Software Requirements*

Constance Heitmeyer, James Kirby, Bruce Labaw and Ramesh Bharadwaj

Naval Research Laboratory, Code 5546, Washington, DC 20375, USA

Abstract. A controversial issue in the formal methods community is the degree to which mathematical sophistication and theorem proving skills should be needed to apply a formal method and its support tools. This paper describes the SCR (Software Cost Reduction) tools, part of a "practical" formal method—a method with a solid mathematical foundation that software developers can apply without theorem proving skills, knowledge of temporal and higher order logics, or consultation with formal methods experts. The SCR method provides a tabular notation for specifying requirements and a set of "light-weight" tools that detect several classes of errors automatically. The method also provides support for more "heavy-duty" tools, such as a model checker. To make model checking feasible, users can automatically apply one or more abstraction methods.

1 Introduction

Given the high frequency of requirements errors, the serious accidents they may cause, and the high cost of correcting them, tools that aid software developers in the early detection of requirements errors are crucial. To be effective, the tools must be usable by software developers on industrial-strength projects and should be based on a formal model of requirements. The formal model provides a solid basis for formal analysis of the specification, which detects many classes of errors automatically.

For a requirements tool to be useful to software developers, the tool must be part of a development method that provides guidance on those decisions the requirements specification should record and those it should not (i.e., the method distinguishes requirements decisions from design decisions) and guidance on making, evaluating, and recording the decisions. The development method should also provide notations that software developers can apply easily in constructing a requirements specification. Finally, the method should not require the developers to be experts in the formal model underlying the tool.

The SCR (Software Cost Reduction) requirements method is a formal method based on tables for specifying the requirements of safety-critical software systems. Designed for use by engineers, the method has been applied to a variety of practical systems, including avionics systems, telephone networks, and nuclear power plants. Originally formulated by NRL researchers to document the

* This work was supported by the Office of Naval Research and SPAWAR.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
1998		**00-00-1998 to 00-00-1998**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
SCR*: A Toolset for Specifying and Analyzing Software Requirements	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Naval Research Laboratory,Code 5546,4555 Overlook Avenue, SW,Washington,DC,20375**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**7**	

requirements of the Operational Flight Program (OFP) of the US Navy's A-7 aircraft [11, 1], SCR has been used in practice by a number of industrial organizations, such as Grumann, Bell Laboratories, Ontario Hydro, and Lockheed, to specify software requirements. For example, in 1993-94, Lockheed used SCR tables to specify the complete requirements of the C-130J OFP [5], a program containing more than 230K lines of Ada code.

Introduced in 1995 [8, 9], SCR* is an integrated suite of tools supporting the SCR requirements method. Figure 1 illustrates SCR*, which includes a *specification editor* for creating a requirements specification, a *dependency graph browser* for displaying the variable dependencies in the specification, a *consistency checker* for detecting well-formedness errors (e.g., type errors and missing cases), a *simulator* for validating the specification, and a *model checker* for checking application properties. Currently, more than 50 organizations in the US, Canada, UK, and Germany, including industrial and government organizations as well as universities, are experimenting with SCR*.

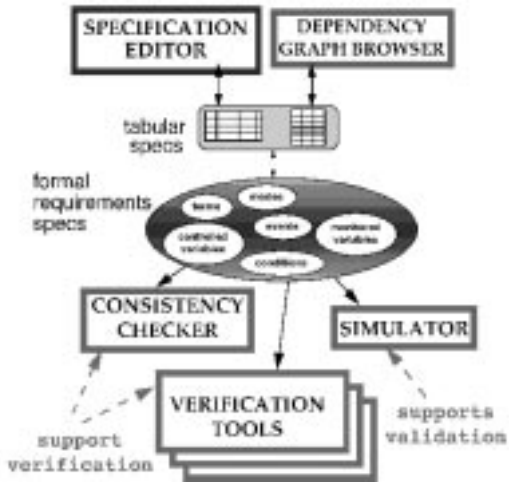

Fig. 1. SCR*: Tools supporting the SCR requirements method

To date, SCR* has been applied successfully in three external pilot projects. In the first, researchers at NASA's IV&V Facility used SCR* to detect missing cases and nondeterminism in the prose requirements specification of software for the International Space Station [4]. In the second project, engineers at Rockwell-Collins used SCR* to expose 24 errors, many of them serious, in the requirements specification of an example flight guidance system [14]. Of the detected errors, a third were uncovered in constructing the specification, a third in running the consistency checker, and the remaining third in executing the specification with the simulator. In a third project, researchers at the JPL (Jet Propulsion Laboratory) used SCR* to analyze specifications of two components of NASA's Deep Space-1 spacecraft for errors [13].

In a fourth pilot project, NRL applied the SCR tools, including a newly integrated model checker [3], to a sizable contractor-produced requirements spec-

ification of the Weapons Control Panel (WCP) for a safety-critical US military system [10]. The tools uncovered numerous errors in the contractor specification, including a serious safety violation. Translating the contractor specification into the SCR tabular notation, using SCR* to detect specification errors, and building a working prototype of the WCP required only one person-month, thus demonstrating the utility and cost-effectiveness of the SCR method.

2 The SCR Requirements Model

An SCR requirements specification describes the required system behavior as the composition of a nondeterministic environment and a (usually) deterministic system [7]. The system environment contains *monitored* and *controlled quantities*, quantities that the system monitors and controls. The environment nondeterministically produces a sequence of input events, where an *input event* is a change in some monitored quantity. Beginning in some initial state, the system responds to each input event in turn by changing state and possibly changing one or more controlled quantities. In SCR, the system behavior is assumed to be *synchronous*—the system completely processes one input event before processing the next input event.

The SCR formal model, a special form of the classic state machine model, represents a system Σ as a 4-tuple, $\Sigma = (S, S_0, E^m, T)$, where S is a set of states, $S_0 \subseteq S$ is the initial state set, E^m is the set of input events, and T is the transform describing the allowed state transitions [7]. In the formal model presented in [7], the transform T is deterministic, a composition of smaller functions called *table functions*, derived from the tables in an SCR specification. The formal model requires the information in each table to satisfy certain properties. These properties guarantee that each table describes a total function.

In SCR, two relations, NAT and REQ, describe the required system behavior. NAT specifies the natural constraints on the system behavior—constraints imposed by physical laws and the system environment. REQ specifies the relation that the system must enforce between the monitored and controlled quantities. To specify REQ concisely, the SCR method uses mode classes, conditions, and events. A *mode class* organizes the system states into equivalence classes, each called a *mode*. The SCR model includes a set RF containing the names of all variables (e.g., monitored and controlled variables, mode classes) in a given specification and a function mapping each variable in RF to a set of values. In the model, a *state* is a function mapping each variable in RF to its value, a *condition* is a predicate defined on a system state, and an *event* is a predicate defined on two system states when any state variable changes.

3 The SCR Tools

Specification Editor. To create, modify, or display a requirements specification, the user invokes the specification editor [8]. Each SCR specification is organized into dictionaries and tables. The dictionaries define the static information in the specification, such as the names and values of variables and constants, the user-defined types, etc. The tables specify how the variables change in response

to input events. One important class of tables specifies the behavior of controlled variables.

Dependency Graph Browser. Understanding the relationship between different parts of a large specification can be difficult. To address this problem, the Dependency Graph Browser (DGB) represents the dependencies among the variables in a given SCR specification as a directed graph [9]. By examining this graph, a user can detect errors such as undefined variables and circular definitions. The user can also use the DGB to display and extract subsets of the dependency graph, e.g., the subgraph containing all variables upon which a selected controlled variable depends.

Consistency Checker. The consistency checker [7, 9] analyzes a specification for properties derived from the SCR requirements model. It exposes syntax and type errors, variable name discrepancies, missing cases, unwanted nondeterminism, and circular definitions. When an error is detected, the consistency checker provides detailed feedback to facilitate error correction. A form of static analysis, consistency checking is performed without execution of the specification or a reachability analysis and is hence more efficient than model checking. In developing an SCR specification, the user normally invokes the consistency checker first and postpones more heavy-duty analysis such as model checking until later. By exploiting the special properties guaranteed by consistency checking (e.g., determinism), later analyses can be more efficient [3].

Simulator. To validate a specification, the user can run the simulator [9] and analyze the results to ensure that the specification captures the intended behavior. Additionally, the user can define invariant properties believed to be true of the required behavior and, using simulation, execute a series of scenarios to determine if any violate the invariants. To provide input to the simulator, the user either enters a sequence of input events or loads a previously stored scenario.

The simulator supports the construction of front-ends, tailored to particular application domains. One example is a customized front-end for pilots to use in evaluating an attack aircraft specification (see Figure 2). Rather than clicking on monitored variable names, entering values for them, and seeing the results of simulation presented as variable values, a pilot clicks on visual representations of cockpit controls and sees results presented on a simulated cockpit display. This front-end allows the pilot to move out of the world of requirements specification and into the world of attack aircraft, where he is the expert. Such an interface facilitates customer validation of the specification. A second customized front-end, part of the WCP prototype mentioned above, has also been developed.

Model Checker. Recently, the explicit state model checker Spin [12] was integrated into SCR* [3]. After using SCR* to develop a formal requirements specification, a developer can obtain an automatic translation of the specification into *Promela*, the language of Spin, and then invoke Spin within the toolset to check properties of the specification. Currently, the model checker analyzes invariant properties. The user can use the simulator to demonstrate and validate any property violation detected by Spin.

Fig. 2. Customized simulator front-end for an attack aircraft specification

The number of reachable states in a state machine model of real-world software is usually very large, sometimes infinite. To make model checking practical, we have developed sound methods for deriving abstractions from SCR specifications [3]. The methods are practical: none requires ingenuity on the user's part, and each derives a smaller, more abstract model automatically. Based on the property to be analyzed, these methods eliminate irrelevant variables as well as unneeded detail from the specification. For example, prior to invoking Spin to check the WCP specification for a safety property, we used our abstraction methods to automatically reduce the number of variables from 258 to 55 and to replace several real-valued variables with finite-valued variables, thus making model checking feasible [10].

4 Comparison with Other Tools

The method most closely related to SCR is the Requirements State Machine Language (RSML) and associated tools [6]. In [2], Anderson et al. describe the use of the model checker SMV to analyze a component of the TCAS-II specification expressed in RSML. Unlike our approach to limiting state explosion which reduces the specification by applying sound abstraction methods, Anderson et al. propose a more efficient encoding for the BDD representation of the

RSML specification. More recently, Park et al. [15] have used the Stanford Validity Checker (SVC) to check the consistency of RSML specifications. Their approach is similar to that used by the consistency checker in SCR* [7, 9].

SCR* can be distinguished in three major ways from other tools. First, unlike most commercial tools for requirements specification, SCR* has a solid mathematical foundation, thus allowing sophisticated analyses, such as consistency checking and model checking, largely unsupported by current tools. Second, the SCR tools, unlike most research tools, have a well designed user interface, are integrated to work together, and provide detailed feedback when errors are detected to facilitate their correction. Finally, users of SCR* can do considerable analysis *without* interaction with application experts or formal methods researchers, thereby providing formal methods usage at low cost.

References

1. T. A. Alspaugh et al. Software requirements for the A-7 aircraft. Report 9194, Naval Research Lab, Wash. DC, 1992.
2. R. J. Anderson et al. "Model checking large software specifications." *Proc. 4^{th} ACM SIGSOFT Symp. Foundations of Software Eng.*, October 1996.
3. R. Bharadwaj and C. Heitmeyer. "Model checking complete requirements specifications using abstraction." *Journal of Automated Software Eng.* (to appear).
4. S. Easterbrook and J. Callahan. "Formal methods for verification and validation of partial specifications: A case study." *Journal of Systems and Software*, 1997.
5. S. Faulk et al. "Experience applying the CoRE method to the Lockheed C-130J." *Proc. 9^{th} Annual Computer Assurance Conf. (COMPASS '94)*, June 1994.
6. M. P. E. Heimdahl and N. Leveson. "Completeness and consistency analysis of state-based requirements." *Proc. 17^{th} Int'l Conf. on Software Eng. (ICSE'95)*, Seattle, WA, Apr. 1995.
7. C. Heitmeyer, R. Jeffords, and B. Labaw. "Automated consistency checking of requirements specifications." *ACM Trans. Software Eng. and Method.* 5(3), 1996.
8. C. Heitmeyer et al. "SCR*: A toolset for specifying and analyzing requirements." *Proc. 10^{th} Annual Conf. on Computer Assurance (COMPASS '95)*, June 1995.
9. C. Heitmeyer, J. Kirby, and B. Labaw. "Tools for formal specification, verification, and validation of requirements." *Proc. 12^{th} Annual Conf. on Computer Assurance (COMPASS '97)*, June 1997.
10. C. Heitmeyer, J. Kirby, and B. Labaw. "Applying the SCR requirements method to a weapons control panel: An experience report." *Proc. 2^{nd} Workshop on Formal Methods in Software Practice (FMSP'98)*, St. Petersburg, FL, March 1998.
11. K. L. Heninger. Specifying software requirements for complex systems: New techniques and their application. *IEEE Trans. on Software Eng.* SE-6(1), Jan. 1980.
12. G. J. Holzmann. *Design and Validation of Computer Protocols.* Prentice-Hall, 1991.
13. R. R. Lutz and H.-Y. Shaw. "Applying the SCR* requirements toolset to DS-1 fault protection." Report D15198, Jet Propulsion Lab, Pasadena, CA, Dec. 1997.
14. S. Miller. "Specifying the mode logic of a flight guidance system in CoRE and SCR." *Proc. 2^{nd} Workshop on Formal Methods in Software Practice (FMSP'98)*, St. Petersburg, FL, March 1998.
15. D. Y. W. Park et al. "Checking properties of safety-critical specifications using efficient decision procedures." *Proc. 2^{nd} Workshop on Formal Methods in Software Practice (FMSP'98)*, St. Petersburg, FL, March 1998.

This article was processed using the LaTeX macro package with LLNCS style